To take part in the action...

- Ensure you have a working Erlang installation
- Grab my snapshot of Erlyvideo:
  - `git clone https://github.com/aronisstav/erlyvideo.git`
- Build it (after possibly changing rebar.config)
  - `make`
- Invoke this curious command:
  - `dialyzer --output_plt my_plt --build_plt --apps erts kernel stdlib`

How to start using Dialyzer in your project

Stavros Aronis

dialyzer@softlab.ntua.gr

Outline

- Why should you be using it?
- How to set it up and run it?
- How to handle warnings reported by it?
- New features available soon!

Why should you be using it?

- No modifications in your code required
- Can detect discrepancies early
- Can check the consistency between the documentation and the implementation
- Is really mature and constantly improving
- You are being watched! (http://dialyzer.softlab.ntua.gr)
- Is **never** wrong!

Setting it up

- `Make` things easy
- What to analyze and how to prepare
- The Persistent Lookup Table
- How to keep track of existing vs new warnings

`Make` things easy

- Dialyzer can be run as another “test”:
  - `make dialyzer`
- Should be able to keep track of:
  - Actual changes in your code (is a re-run required?)
  - The Persistent Lookup Table
  - Existing warnings vs new ones

What to analyze?

- All your “actual” code.
- **your:**
  - External applications
    - shouldn't be analyzed
    - Belong to the PLT
- **“actual”:**
  - “Testing” code will produce warnings
  - Instead of filtering these out, avoid them in the first place

How to prepare your code?

- Dialyzer needs access to the source code
- … but analysis from source code requires:
  - Included files to be added explicitly
  - Parse transformations to be in the code path
- … just like the compiler!
- Use compiled modules: .beam files as input
- … compiled with **`+debug_info`**

The Persistent Lookup Table

- Your application **will** call OTP functions.
- You don't need to re-analyze these every time!
- The same applies to any other “external” application
- The Persistent Lookup Table (PLT) can store results of the analysis of these modules and consult them when finding calls to them
- **`dialyzer --output_plt my_plt --build_plt --apps erts kernel stdlib`**

Existing vs. new warnings

- When initially run, Dialyzer might report some warnings
- Fix them at your own pace...
- … keeping track of them so:
  - You record your progress
  - You do not introduce new discrepancies
- Do NOT add specs/types before fixing existing warnings!

ACTION!

- If you have prepared Erlyvideo:
  - `git clone https://github.com/aronisstav/erlyvideo.git`
  - `make`
- See this setup in action:
  - `git remote update`
  - `git checkout 7e7db8`
  - `OR: git show 7e7db8`
- Run '`make dialyzer`'
- To speed this up: kill it, copy `my_plt` to `test/dialyzer/plt` and run it again!

How to handle warnings?

- We got a long list of warnings
- How to actually debug a warning?
- Where to begin?

Debugging warnings

- Try to minimize the modules that produce the warning.
- Beginning with the module that includes the warning...
- … run:
  - `make; dialyzer ebin/buggy.beam`
- … if it doesn't show up add some of the unknown modules.
- When you can get it it's time for action!

The call will never return...

“The call to <module>:<function>(<Args with types>)

- *will never return”*
- *does not have opaque terms ...”*

Can be fixed by:

- Checking the documentation
- Respecting opaque types
- Correcting a possibly wrong spec

The call will never return...

- OTP documentation related:
  - Such examples are `filename:join` calls with atoms as arguments and `file:open` calls with a single atom denoting “mode” instead of an option list. In these cases you should consult the documentation and adhere to it.
- Investigating dubious specs
  - run Dialyzer with the `--no_spec` flag to see if the problems disappear. If this is the case you should fix the specifications.

Function has no local return

- *“Function <function>/<arity> has no local return”*
  - Usually eliminated along with the failing calls
  - If not, you might have to follow a chain of calls
  - (A function with no local return will often be the reason for an identical warning in any function that calls it).

Record construction violates the declared type of field(s)

- *“Record construction <record> violates the declared type of field(s) <field>::<type>”*
  - Comment out the types of the record’s fields
  - re-introduce them in an incremental way, adding any missing values to the type.

More information

- Initial submission was a ~15 page guide with more information on:
  - Tricks to analyze faster (aka “enable HiPE”)
  - Common causes for Dialyzer crashes
  - Usage of TypEr during debugging
  - More details on other warnings
  - General advice on modernizing specs/types
- Soon to be available on Dialyzer's site
- Already available by e-mail :-)

Dialyzer's development

- Behaviour usage analysis
  - Appropriate implementation of callbacks
  - Makes use of the new “-callback” Erlang attribute, used to specify a behaviour's callbacks
  - To be included in R15
- Stronger “success typing” inference
  - Keeping relations between arguments/results
- More concurrency error detection
  - “Lost” messages, deadlocks

Thank you!